A SOFTWARE SYSTEM FOR OPTIMIZATION BASED INTERACTIVE COMPUTER-AIDED DESIGN

by

M. A. Bhatti, T. Essebo, W. Nye, K. S. Pister
E. Polak, A. Sangiovanni-Vincentelli and A. Tits

Memorandum No. UCB/ERL M80/14

11 April 1980

A SOFTWARE SYSTEM FOR OPTIMIZATION BASED INTERACTIVE COMPUTER-AIDED DESIGN

by

M. A. Bhatti, T. Essebo, W. Nye, K. S. Pister
E. Polak, A. Sangiovanni-Vincentelli, and A. Tits

Memorandum No. UCB/ERL M80/14

11 April 1980

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A SOFTWARE SYSTEM FOR OPTIMIZATION BASED INTERACTIVE COMPUTER-AIDED DESIGN

by

M.A. Bhatti [1], T. Essebo [2], W. Nye [3], K.S. Pister [1],
E. Polak [3], A. Sangiovanni-Vincentelli [3], A. Tits [3].

1. THE CASE FOR OPTIMIZATION BASED INTERACTIVE CAD.

The term computer-aided design is used to describe a great variety of activities. In electronics, computer-aided design often amounts to no more than simulation of electronic circuits coupled with a cut-and-try procedure. At first the designer chooses an initial design configuration. Then the configuration is analyzed by means of a computer program which simulates the behavior of the physical system. By looking at the results of the computer simulation, the designer adjusts parameter values in an attempt to satisfy a set of given specifications which are not met by the initial configuration and/or obtain a better design in terms of performances and/or production costs. After the adjustment, a new simulation is performed and the overall procedure is iterated until a satisfactory design is obtained.

Over the last decade research in computer simulation of electronic circuits has made considerable progress resulting in a number of excellent simulation programs (e.g. [1-3]). Since the late 1960's it has been felt that the cut-and-try design mode could and should be improved considerably [4,5]. In a cut-and-try design mode the designer essentially assumes the role of a heuristic master optimization and inequality solving algorithm. Unfortunately, pure heuristics are generally inefficient in searching a multidimensional parameter space and hence they cannot

[1] Department of Civil Engineering
University of California, Berkeley
[2] Department of Automatic Control
Institute of Technology, Lund, Sweden
[3] Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

be relied on to produce a feasible design, let alone an optimal one. Consequently, an efficient design procedure must make substantial use of optimization algorithms so as to relieve the designer from the drudgery of search in the parameter space and to allow him to concentrate on the conceptual aspects of the design.

Despite considerable research activity in computer optimization of electronic circuits (e.g. [6-9]), according to recent surveys [10,11] optimization techniques are not used as widely as might be expected. There are several reasons for this. Perhaps the most important one is that the optimization algorithms used until now have been too primitive for the task at hand. For example, they are not capable of solving non convex problems with tolerances and with tolerances and post-manufacture tuning. Even in the simple cases, with no tolerances involved, the cost/benefit ratio has frequently been unfavorable because the algorithms failed to converge to a solution in a reasonable amount of computer time. This situation may be caused by the ill-conditioning of the mathematical programming problem into which the design problem was translated, by the weak convergence properties of the algorithms used (e.g. penalty function with conjugate gradients as subroutine for unconstrained optimization) or by a poor choice of initial design and/or algorithm parameters. Since any algorithm for optimization based computer aided design requires a number of simulations per iteration and since the cost of simulation ranges from 15 secs for a simple problem to minutes for a realistic design problem (on a CDC 6400), it is clear that slow convergence or no convergence at all may be considered as a very expensive accident!

Recently new algorithms have been developed for design problems involving tolerances [12,14] and tuning [15]. At the same time, methods for early detection of ill-conditioning in the mathematical programming problem into which the design problem was translated, are emerging. Also heuristics are currently being developed which help avoid translation ill-conditioning. Since in general the transcription of a design problem into a mathematical programming problem is not unique, these heuristics suggest ways for changing the transcription of the design problem to eliminate the ill-conditioning. More robust algorithms, i.e. algorithms

with guaranteed convergence properties are now being used [16]. However, these algorithms are still very sensitive to the choice of internal parameters as well as initial values of design parameters.

In our opinion, a new design methodology based on interactive graphic computing is indispensable. Interactive computing permits one to abort, stop and restart or otherwise modify a computation as it progresses, resulting in very substantial savings not only in computing time, but also in the overall time needed to carry out a design. As an example, suppose that an initial design proposed by a designer fails to meet specifications. When using an interactive CAD system, he could identify this fact by observing the computations. He could stop the computation and either modify the structure of his design or experiment with relaxation of the specifications. Next, in the case of ill-conditioning, he could change the description of the design problem into a different mathematical programming problem by observing the heuristic information displayed on the screen. Finally, he would be in an ideal situation to perform trade-offs of one desirable goal to obtain an improvement in another. Since it is obviously impossible either to compute or to display an entire multidimensional trade-off surface, interactive computing techniques are being developed which will enable the designer to find a satisfactory compromise solution on the basis of a sequence of computations which he must guide interactively [17].

One of the goals of our research effort is to develop a software system for optimization based interactive computer-aided design. In the following sections we describe a prototype interactive software system INTEROPTDYN (INTERactive OPTimization of DYNamical systems), built around the language INTRAC [18]. In section 2 we discuss how interaction should be implemented in an optimization package. In section 3 we describe some of the features of INTEROPTDYN.

2. INTERACTIVE GRAPHICS IN OPTIMIZATION BASED CAD

When expressed as an optimization problem most engineering design problems become a nonstandard mathematical programming problem of the form:

$$\text{P1}: \quad \min \{ f(x) \mid g^j(x) \leqq 0,\ j=1,\ldots,q;\ \max_{p^i \in P^i} \varphi^i(x,p^i) \leqq 0,\ i=1,\ldots,m \}$$

where x is the vector of the design parameters. Some of the components of the p^i's are parameters such as temperature, time or frequency, while others are tolerances on the nominal design. The sets $P^i, i=1,\ldots,m$ are generally intervals or n dimensional boxes or other convex sets in R. We assume that the cost and the constraint functions f and $g^j, j=1,\ldots,q$ are continuously differentiable. We also assume that the functions $\varphi^i, i=1,\ldots,m$ are continuously differentiable with respect to x and Lipschitz continuous with respect to p^i. Of course the constraints of the form $\max_{p^i \in P^i} \varphi^i(x,p^i) \leqq 0$ are not continuously differentiable with respect to x. For the sake of simplicity, in the rest of this section we consider design problems where these constraints are not present.

To illustrate the use of interactive graphics in solving such a problem suppose that we use a feasible directions method [19]. Let $F = \{ x \mid g^j(x) \leqq 0,\ j=1,\ldots,q \}$ be the feasible region for the design problem. Given $\varepsilon > 0$, the set $J_\varepsilon(x) = \{ j \in \{1,\ldots,q\} \mid g^j(x) \geqq -\varepsilon \}$ is called the set of indices of the ε-active constraints. Now we introduce a feasible direction algorithm to discuss the use of graphic interaction in optimization.

Feasible Directions Algorithm.

Data $x_0 \in F$.

Parameters $\alpha, \beta \in (0,1)$; $\varepsilon_0 > 0$; $\overline{\varepsilon} \geqq 0$.

Step 0 Set $i=0$.

Step 1 Set $\varepsilon = \varepsilon_0$.

Step 2 (Direction finding subprocedure)

Compute $h_\varepsilon(x_i)$ and $\vartheta_\varepsilon(x_i)$, where

$$h_\varepsilon(x_i) = -\operatorname{argmin} \{ \|h\|^2 \mid h = \lambda_0 \nabla f(x_i) + \sum_{j \in J_\varepsilon(x_i)} \lambda_j \nabla g^j(x_i);$$

$$\sum_{j \in J_\varepsilon(x_i)} \lambda_j = 1;\ \lambda_j \geqq 0, j \in J_\varepsilon(x_i) \}$$

and $\vartheta_\varepsilon(x_i) = \|h_\varepsilon(x_i)\|^2$.

Step 3 (Termination criterion)

If $\varepsilon \leq \bar{\varepsilon}$, compute $\vartheta_0(x_i)$ and stop if $\vartheta_0(x_i) \leq \bar{\varepsilon}$. Else proceed.

Step 4 (ε−reduction)

If $\vartheta_\varepsilon(x_i) \leq \varepsilon$, set $\varepsilon = \frac{\varepsilon}{2}$ and go to Step 2. Else proceed.

Step 5 (Stepsize computation by Armijo rule [19])

Compute the smallest integer $k_i \geq 0$ such that

$$f(x_i+\beta^{k_i}h_\varepsilon(x_i)) - f(x_i) \leq -\alpha\beta^{k_i}\|h_\varepsilon(x_i)\|^2 \tag{2.1a}$$

$$g^j(x_i+\beta^{k_i}h_\varepsilon(x_i)) \leq 0, \quad j=1,\ldots,q. \tag{2.1b}$$

Step 6 Set $x_{i+1} = x_i+\lambda h_\varepsilon(x_i)$, set $i=i+1$ and go to Step 1.

Under reasonably weak conditions on the feasible region $\mathbf{F}$ if $\bar{\varepsilon}=0$, the above algorithm is guaranteed to produce a sequence of design parameters whose accumulation points satisfy a first order necessary optimality condition [16]. Its computational efficiency depends critically upon the values of the parameters. Each of the parameters controls a particular phase of the optimization algorithm. For example, ε and $\bar{\varepsilon}$ control the termination of the algorithm, ε_0 controls which constraints have to be taken into account at the beginning of each iteration of the algorithm in computing the descent direction, α and β control respectively the slope of the line in fig.1 and the rate of reduction of the step size in the Armijo rule. Unfortunately, the optimal values of the parameters are problem dependent. A designer with knowledge of the mechanisms of the algorithm may see from the results of the early iterations that the computation is not progressing satisfactorily. In order to avoid continued inefficient use of the algorithm, the designer must be able to interrupt the computing process to analyse the causes of the unsatisfactory situation, to change the values of the data or of the parameters and restart the computing process. These actions can be accomplished efficiently only by interaction. However, in order to make interaction as effective as possible, the designer must be provided with indicators which can guide him in the detection of poor computational behavior of the algorithm. Moreover, these indicators should provide information on as to how the parameters of the algorithm should be changed so as to improve its performance.

For example, suppose that the algorithm, after computing the search direction $h_\varepsilon(x_i)$, gets hung up in the Armijo step loop because test (2.1b) is not satisfied. This is certainly extremely undesirable, since in CAD problems the required function evaluations need expensive simulations. A designer may try to correct this situation by adjusting some of the parameters of the algorithm or by rescaling the problem. In order to do that, he has to detect which constraint is forcing the decrease of the step size in the Armijo step. Then he must be able to check if this constraint is in the set of the ε−active constraints. If this constraint is not in the set of the ε−active constraints, then the search direction computation does not "see" this constraint and as a consequence the step size may have to be reduced considerably before test (2.1b) is satisfied. In this case the designer could modify ε by increasing it, so that the neglected constraint is in the set of ε−active constraints, if the process is resumed by going back to step 2. When increasing ε the designer must be careful not to make ε so large that too many other constraints become active, because as a result, $h_\varepsilon(x_i)$ may become too small and ε would then be reduced in step 3, wasting several cycles in step 2. Furthermore, gradient computations are costly and should be kept to a minimum. If he finds out that the impeding constraint is in the ε−active constraints set, then he may try a different strategy to improve the performance of the algorithm. In this case, the poor computational behavior may be caused by bad scaling. Let us consider first the geometrical interpretation of the search direction. According to step 2 the search direction direction calculation problem turns out to be the negative of the nearest vector to the origin in the convex hull of the gradients of the cost and of the ε-active constraints. From the geometry of the direction calculation problem, we deduce that if the L_∞ norm of the gradient of the impeding constraint is very large compared to the norms of the gradients of the other ε-active constraints and/or of the cost, then the search direction computed by solving the quadratic programming problem in step 2 may not take into account the gradient of the constraint which is causing difficulty (see fig.2). This situation may be detected by looking at the angle between the search direction and the gradient of the constraint. If this angle is close to 90° then, $h_\varepsilon(x_i)$ does not adequately take into account the impeding constraint. To correct

this situation, the designer could multiply the gradient of the limiting constraint by a "pushfactor" to make the L_∞ norm of this vector comparable to those of the other vectors considered in step 2.

It turns out that in almost all the critical phases of the optimization algorithm similar information should be made available to the designer. Of course, this information can be given numerically at each iteration of the algorithm and at each iteration of the subalgorithms (loops) inside the steps of the algorithm. However, it is easy to see that a massive quantity of numerical data presented on a screen, may overwhelm a designer, jeopardizing the efficiency of the design procedure. A much more efficient method for carrying out interaction is through graphics.

Let us consider the computational problem described above. To help the designer to detect the reason for the poor performance of the algorithm, we could display a bar chart plotting the values of the constraints before entering the Armijo step, side by side with the values of the constraint at each iteration of the Armijo step. By displaying also a line corresponding to zero and a line corresponding to ε, the designer could immediately detect if an impeding constraint, i.e. a constraint with the corresponding bar above the zero line in the Armijo iterations, is not in the set of the ε-active constraints by looking at the bar corresponding to the value of the constraint before the Armijo step is entered (see fig.3). In fact, if this bar is below the ε-line, then the constraint is not in the set of the ε-active constraints. The use of color graphics could further improve man-machine interaction. In the example described above, if the bars corresponding to constraints which are not satisfied are plotted in red, the bars corresponding to the constraints which are satisfied are plotted in green, the ε-line in yellow, the designer could grasp all the information he needs at a glance. The need for graphical display of information is even stronger, if we consider a design problem with distributed constraints (i.e. constraints of the max form). In this case the designer needs to have a feeling for the change in the function $\varphi^i(x,p^i)$ when x changes. While the plot of the function $\varphi^i(x,p^i)$ with respect to p^i for a fixed x gives information easily grasped by a designer, the numerical data printed on a screen cannot

be absorbed without a lengthy analysis.

We noticed before that the designer may need to perform additional computations in order to gather relevant information to rescale the problem. It seems undesirable to load the main algorithm with all kinds of side computations to cover all the possible needs of the designer, partly because it is not possible to anticipate all such needs and partly because the designer may come up with unforeseen tests which are particularly efficient for his problem. Thus, an optimization based computer-aided design system should permit improvised side computations on variables, vectors and matrices used in the optimization algorithms. Consequently, the system should incorporate a powerful scratchpad, capable of matrix operations such as inversions, transpositions and calculation of condition numbers.

As a result, we designed a prototype system with the following criteria in mind:

1- Ease of interaction should be emphasized. A designer should be able to interrupt the computing process, change the parameter values and restart the process. Moreover he should be able to control the flow of the algorithm by single stepping through its loops. (This feature is most useful in diagnosing where the computation jammed up and what is the probable cause of the jamming of the algorithm).

2- Graphical display of quantities computed by the optimization and the simulation algorithms should be possible. Color graphics should be used to enhance man-machine interaction.

3- A powerful, high level, scratchpad for side computations on variables, vectors and matrices used in the optimization algorithm should be available.

3. THE INTEROPTDYN SYSTEM

The INTEROPTDYN system is an experimental interactive software package for optimization based computer-aided design of dynamical systems developed at the University of California, Berkeley by a team formed by the authors of this paper. The system is running on a DEC VAX 11/780 computer granted by NSF for research on interactive computer-aided design of

engineering systems. The operating system is a virtual memory version of UNIX developed at the University of California, Berkeley,(UNIX is a Bell system trade mark). The system can be used to solve design problems of the form P1 where the P^i's are intervals. At present, the system consists of :

1- A main program (OPTDYN) written in FORTRAN, implementing the Gonzaga-Polak-Trahan phase I-phase II method of feasible directions [20,21].

2- The interpreter of an interactive language, INTRAC-C, evolved from INTRAC, an elementary interactive language, originally developed at the Department of Automatic Control, Lund Institute of Technology, Sweden [19]. INTRAC-C is written in FORTRAN and produces FORTRAN as intermediate code. We shall refer to the interactive language interpreted by INTRAC-C as the INTRAC-C language.

3- A set of procedures (macros) written in the INTRAC-C language.

The heart of the system is INTRAC-C. It is an application specific extension of INTRAC, which is an elementary interaction language conceived in such a way that applications specific extensions are easy to construct. The INTRAC-C language has four sets of problem independent commands:

1- The original INTRAC commands for assignment of variables, conditional and unconditional branching, looping, input and output.

2- Commands for interacting with the optimization package.

3- Graphics commands.

4- Scratchpad commands, i.e. powerful commands for algebraic manipulations of scalars, vectors and matrices.

INTRAC-C allows the use of macros (procedures). A macro is implemented in INTRAC-C as a text file. When a macro is called by the user, INTRAC-C reads the file corresponding to the macro from mass storage and takes the appropriate action. INTEROPTDYN has a simple text editor to modify macros.

Variables in the INTRAC-C language can be local or global. Local variables are local to the macro level and are defined when they are first given a value in a read statement or in an assignment statement. Global variables are always accessible and may pass information between macros.

A feature which makes INTRAC-C particularly useful in interactive CAD is the possibility of suspending the execution of a macro by using the command SUSPEND. The execution of the macro can be resumed by using the command RESUME. When a macro is suspended, commands can be inputted from the terminal. A typical use of this feature would be the following. When executing a macro, the program may need some information from the user to perform effectively its task. Then the macro is suspended and a question-answering phase begins. In this phase, all the variables of the macro are accessible and the user can change values of variables local to the macro. When the interaction is ended, the macro is resumed and the computation progresses. In INTEROPTDYN it is also possible to interrupt the execution of the macro *externally*, forcing it in suspended mode. This feature allows the user to abort an unsatisfactory run and to access parameters and variables in the program outside a fixed frame. Parameters and variables of the optimization algorithm which need to be changed interactively or which need to be accessible to INTRAC-C must be deposited in the symbol table of INTEROPTDYN.

Among the application commands available in INTRAC-C we find the commands which handle the interaction with OPTDYN. The first step in using these commands was to decide where interaction should take place. According to the considerations of section 2, interaction should be implemented at each step of the main loop of the algorithm as well as at each step of every internal loop. Thus breakpoints have been inserted after the corresponding statement of OPTDYN. At each breakpoint a subroutine, INTCAL, is called. This subroutine checks the condition associated with the break point. The condition may assume the following values:

1- NEVER: In this case no action is taken and the control is returned to the main program.

2- ALWAYS: In this case INTRAC-C is called and an interaction phase takes place.

The condition of a breakpoint can be changed by the HALT command of INTRAC-C. An INTRAC-C command has the general form :

< command identifier> < argument list>

The following notation will be used in describing the syntax of commands:

1- < > denotes that the enclosed term is not used literally but is replaced by its appropriate value.

2- { } groups terms together.

3- [ ] groups terms together and denotes that the group is optional.

The HALT command has the following structure: HALT < breakpoint> < condition> , where < breakpoint> is the name of the breakpoint where the condition is to be set, and < condition> can be ALWAYS, NEVER or an IF-clause followed by ALWAYS or NEVER. The IF-clause is used to change the condition dynamically. For example, the command HALT ARMIJO IF ITER> 3 ALWAYS, sets the condition of the breakpoint ARMIJO to ALWAYS if the number of iterations in the Armijo step is larger than 3.

A number of other commands are available for the control of flow of INTEROPTDYN. For the sake of brevity we shall not describe them here.

The variables in the symbol table of INTEROPTDYN can be changed by using the following commands:

1- SET: The SET command has the following structure: SET< variable> = < argument> , where < argument> can be either < variable> or < number> .

2- SETDIM: The SETDIM command changes the dimension of a variable in the symbol table. Its syntax is SETDIM{ncol|nrow} (< variable>)= < argument> , where < ncol> and < nrow> are respectively the column and the row dimension of the variable.

The graphics commands of INTRAC-C can be grouped in two parts:

1- Low level primitives for vector generation, initialization, terminal control, text output, positioning and windowing.

2- High level display functions.

These commands can be executed on the following graphics interactive terminals: Tektronix 4027, Ramtek Micrographics, HP 2648. The first two terminals are color graphics terminals. Our research group has access to 1 Tektronix, 1 Ramtek and 5 HP 2648 terminals. About 15 low level primitives are available. For the sake of brevity, we are going to examine only one of these; the VECTOR command. It is used to draw a vector between two points. Its syntax is : VECTOR < x1> < y1> < x2> < y2> , where < x1> and< x2> are the x-coordinates of the two points and < y1> and < y2> are the y-coordinates of the two points.

Two high level display commands are available; a CURVE and a BAR command. Both commands have the same syntax. The syntax of the commands is < command> < array> < ymin> < ymax> [< topcolor>] [< botcolor> < threshold>] where < array> contains the name of the array carrying the information to be displayed, < ymin> and < ymax> are used for the y-axis scaling, the optional < topcolor> specifies the color to be used in the output if the second option is not used. If the optional < botcolor> is given, then a numeric < threshold> must follow. All entries in the < array> above the < threshold> will appear in the < topcolor> while all entries below the < threshold> will appear in the < botcolor> . The CURVE command plots all the entries in the < array> , while the BAR command produces a barchart. These commands implement among others, the ideas discussed in section 2 for using color graphics to plot indicators for the behavior of the optimization algorithm.

In addition to the main, or INTEROPTDYN, symbol table, there is a second, or scratchpad, symbol table. This symbol table serves to protect the main symbol table as well as for results of side computations. Thus the INTRAC-C set of scratchpad commands can access both symbol tables, but can only alter values in the scratchpad symbol table. The most interesting commands of the scratchpad are:

1- PDIM: This command creates arrays in the symbol table. Its syntax is: PDIM < array> [(< nrow> [:< ncol>])] < type> where < array> is the name of the variable which is being created, < nrow> and < ncol> are the row dimension and the column dimension of the array which can be given optionally. If < nrow> and < ncol> are not given then the variable being created is a scalar. If < ncol> is not given then the variable being created is a column vector and so on. < type> indicates which type is to be attached to the variable. The scratchpad set of commands accepts four types, namely, integer, real, double precision real, and complex.

2- PMAT: This command is used to perform mathematical operations on arrays. It takes two forms:

   a- PMAT< array> = {< array> < number> } < op> < array> , where the first < array> is the name of the array where the result of the operation is stored, the second is the first operand, the third is the second operand and < op> is one of the following matrix operations: *(multiplication), + (addition), -(subtraction)

   b- PMAT< array> = < func> < array> , where < func> can be: INV(inversion), TRANS(transposition), TRACE(trace) or DET(determinant).

3- PSCAL: This command is used to perform mathematical operations on the scalars. Its syntax is similar to PMAT except for the < array> which now is replaced by < scalar> . The operations available are the four basic operations and the functions available are the functions allowed in FORTRAN.

It is obvious that this set of commands meets the specifications indicated in section 2. For example, in step 2 of the Gonzaga-Polak-Trahan algorithm, a quadratic programming problem must be solved to find the search direction. We use the Wolfe algorithm to solve this mathematical programming problem. It is very important that the matrix of the linear constraints be well conditioned for the algorithm to produce a meaningful solution. If the search direction is not satisfactory, the designer could check on the conditioning of this matrix. To do so, he could form a square matrix by multiplying the matrix by its transpose using the functions

and the matrix operations provided in the scratchpad set of commands (a transposition followed by the multiplication of two matrices). Then he could compute the condition number of this square matrix by using the COND function provided in the PMAT command.

Finally, the last component of the INTEROPTDYN system is a set of macros written in the INTRAC-C language which have been found to be sufficiently useful to warrant depositing them in our library. These macros can be divided into three groups:

1- Macros which manage the execution of the optimization algorithm.

2- Macros which implement high level display functions.

3- Macros which make the use of the scratchpad feature easier.

The main macro of the first set is called RUN. This macro enables the execution of a specified number of overall iterations of the optimization algorithm. Its syntax is RUN < nitn> [< display>], where nitn is an integer indicating the number of iterations one wants to perform and < display> is the name of a macro which can be coupled to RUN. This macro will be executed at each iteration of the overall algorithm. The program will stop after the number of iterations specified has been reached (of course, the program may stop before if the optimal design is reached). When the program stops, the macro displays on the screen a set of questions indicating to the user possible changes of algorithm parameters before running more iterations. It is very useful to combine RUN with a display macro which prints or displays graphically the values of the cost and of the constraints while the computation is progressing. In fact, on the basis of the information displayed on the screen the designer may decide to suspend the execution of the macro RUN and to perform side computations or change the values of a few parameters via the SET command.

Several macros are available which implement high level display functions. For example, the macro GRAPH is used to plot the values of an array. Its syntax is GRAPH < array> < color> < mark> [< index1> < index2>] where < array> is the name of the column vector whose entries have to be plotted, < color> is the name of the color to be used when plotting the array and < mark> can assume either the value yes or no. In the first case, the points of

the graph corresponding to the entries of the array are marked with a small asterisk. The optional < index1> and < index2> are used to plot only a subset of entries of the array, namely the ones between the elements with index equal to index1 and with index equal to index2. The macro computes the scale factors to fit the curve in a given window on the screen and it clips out the subarray defined by the indices. Thus, the option can be used to zoom in on a part of the graph which looks particularly interesting to the designer. Graphic macros are built hierarchically so that macros at higher levels call macros at lower levels. GRAPH is a macro at an intermediate level. For example, PLTROW is a macro at higher level which plots a specific row of an array and which calls GRAPH. Its syntax is: PLTROW < array(I:)> < color> [< mark>] where < array(I:)> is the name of a row to be plotted, < color> is the name of the color to be used to plot the array, and the optional < mark> indicates if the coordinate points forming the graph has to be marked. To give an example of how a macro is written in INTEROPTDYN we list PLTROW below.

```
MACRO PLTROW H(I:) C; YESNO
```

The array is H(I:), a row vector extracted from a matrix H by picking up the I-th row. C is the local variable indicating the color of the plot, the ; separates the compulsory arguments from the optional ones, YESNO is the argument which indicates if the plot has to be marked or not.

```
DEFAULT YESNO = N
```

This line of code indicates the default value of YESNO which is no.

```
ROW oH = H(I:)
```

This line uses the macro ROW to create a new variable, oH, which is a row vector.

```
TRS oHT = TRANS(oH)
```

This line uses the macro TRS to create a column vector which is obtained by transposing the row vector of the previous statement.

```
PREM oH
```

This line of code is needed to remove the variable oH which has been created in the scratchpad symbol table by the previous command.

GRAPH oHT C YESNO

This line of code calls the macro GRAPH whose arguments are: the column vector created in the previous statements, the color indicated in the arguments of the macro PLTROW and the variable needed to determine if the plot has to be marked.

PREM oHT

Once we have used oHT we need to remove this variable from the symbol table of the scratchpad so as not to waste memory.

END

This command indicates the end of a macro.

Macros are relatively easy to write but they are inefficient. In fact, since the commands of a macro are interpreted every time a command is read, the command is parsed and executed. Therefore, macros involving loops take a long time to run. On the other hand, the commands are implemented by FORTRAN routines, are compiled and are therefore much more efficient. However they are more difficult to write than macros.

The last set of macros available in INTEROPTDYN make the scratchpad commands easier to use. For example, the macro MM makes the use of the matrix multiplication command much easier. When using the command PMAT A = B*C, we need to create the variable A first, declaring its proper dimensions. MM creates the variable with the right dimensions automatically. Its syntax is: MM < array> = < array> * < array> where the first < array> indicates the name of the array where the product will be stored, the second and the third < array> indicate the name of the arrays to be multiplied. We need not declare the first < array> nor its dimensions.

4. CONCLUSIONS

INTEROPTDYN is a prototype software package for optimization based computer-aided design of engineering systems. Its main features are :

1- Ease of interaction with the optimization algorithm implemented in the package.

2- Extensive use of color graphics.

3- A scratchpad subsystem used to perform side computations needed to monitor the

behavior of the optimization algorithm and to improve it when unsatisfactory.

We are currently incorporating into our system a number of powerful simulation packages which can be called by INTEROPTDYN for function and derivative computation. As a result, we are obtaining a number of very powerful optimization based CAD packages for use in different engineering fields. In parallel to this activity, we are working on a more advanced version of INTEROPTDYN based on a new interactive language which should be more powerful than the INTRAC-C language.

5. ACKNOWLEDGEMENT

Research sponsored by the National Science Foundation under Grants ENG-7810442, PFR-7908261 (RANN), ECS-7913148 and the Joint Services Electronics Program Contract F49620-79-C-0178.

REFERENCES

[1] "Advanced Statistical Analysis Program(ASTAP)", IBM Corp-Data Processing Division, White Plains, NY, Installed User Program, SH 20-1118-X

[2] Nagel, L.W. and D.O.Pederson, "SPICE2: A computer Program to Simulate Semiconductor Circuits"ERL Memo N.ERL-M520, Electronics Research Laboratory, University of California, Berkeley, May 1975.

[3] Becker, D. et al. "SCEPTRE(Improved)", USAF Technical Report AFWL-TR-73-75, Air Force Weapons Laboratory, Kirkland AFB, N.M.1973.

[4] Director, S.W. and R.Rohrer, "The Generalized Adjoint Network and Network Sensitivities" IEEE Trans. Circuit Theory, CT-16, pp.318-323, Aug.1969.

[5] Director, S.W. and R.Rohrer, "Automated Network Design: The Frequency-Domain Case", IEEE Trans. Circuit Theory, CT-16, pp.330-336, Aug.1969.

[6] BandlerJ.W., P.C.Liu and H.Tromp, "A Nonlinear Programming Approach to Optimal Design Centering Tolerancing and Tuning", IEEE Trans. on Circuits and Systems, CAS-23, pp.155-165, Mar.76.

[7] Director, S.W. and G.Hachtel, "The Simplicial Approximation to Design Centering", IEEE Trans. on Circuits and Systems, CAS-24, pp.363-372, July 1977.

[8] Madsen, K., H.Schjaer-Jacobsen, and J.Voldby, "Automated Minimax Design of Networks", IEEE Trans. on Circuits and Systems, CAS-23, pp.456-460, July 1976.

[9] Hachtel, G.D., M.R.Lightner and H.J.Kelly, "Application of the Optimization Program AOP to the Design of Memory Circuits", IEEE Trans. on Circuits and Systems, CAS-22, pp.496-503, June 1975.

[10] Kaplan, G. "Computer-Aided Design", IEEE Spectrum, pp.40-47, Oct.1975.

[11] Pinel, J.F. et al. 'The Impact of Optimization in Network Design" IEEE Proc.1976 Int. Symp. on Circ. and Syst., pp.783-786, 1976.

[12] Schjaer-Jacobsen H. and K.Madsen, "Algorithms for Worst-Case Tolerance Optimization", IEEE Trans. on Circuits and Systems, CAS-26, pp.775-784, Sept.1979.

[13] Brayton, R.K. et al. "A New Algorithm for Statistical Circuit Design Based on Quasi-Newton Methods and Function Splitting", IEEE Trans. on Circuits and Systems, CAS-26, pp.784-795, Sept.1979.

[14] Gonzaga, C. and E.Polak, "On Constraint Dropping Schemes and Optimality Functions for a Class of Outer Approximations Algorithms", SIAM J.Control and Optimization, vol.17, pp.477-493, July 1979.

[15] Polak, E. and A.Sangiovanni-Vincentelli, 'Theoretical and Computational Aspects of the Optimal Design Centering, Tolerancing and Tuning", IEEE Trans. on Circuits and Systems, CAS-26, pp.795-813, Sept.1979.

[16] Polak, E."Algorithms for a Class of Computer-Aided Design Problems: A Review", Automatica, vol.15, pp.531-538, 1979.

[17] Payne, A.N. and E.Polak, "An Interactive Rectangle Elimination Method for biobjective decision making",,IEEE Trans. on Automatic Control, in press.

[18] Wieslander, J. and H.Elmqvist, "INTRAC, A Communication Module for Interactive Programs", Department of Automatic Control Memo LUTFD2/(TFRT-3149)/1-060/1978, Lund Institute of Technology, Lund, Sweden, Aug.1978.

[19] Polak, E. 'Computational Methods in Optimization", Academic Press, NY, 1971.

[20] Gonzaga, C. ,E.Polak and R.Trahan" An Improved Algorithm for a Class of Optimization Problems with Functional Inequality Constraints" Electronics Research Laboratory Memo N.UCB/ERL-M78/56, University of California, Berkeley, 1978

[21] Bhatti, M.A., E. Polak and K.S. Pister "OPTDYN- A General Purpose Optimization Program for Problems with or without Dynamic Constraints" Report No. UCB/EERC-79/16, Earthquake Engineering Research Center, University of California, Berkeley, July 1979.

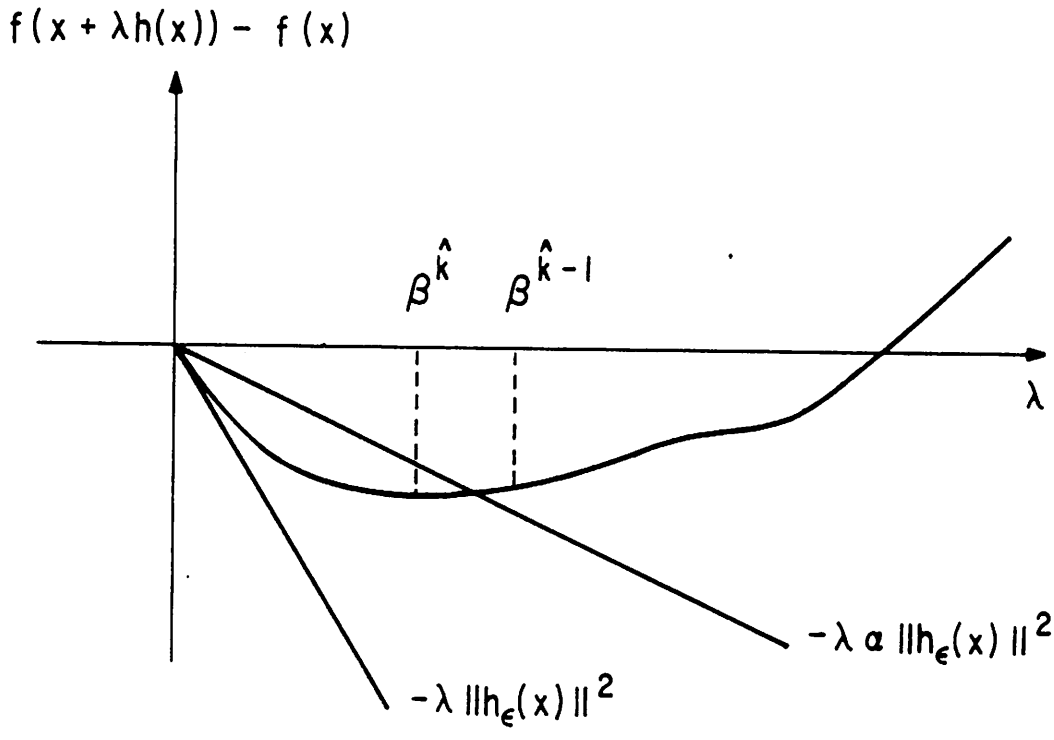

Figure 1. The Armijo step size calculation.

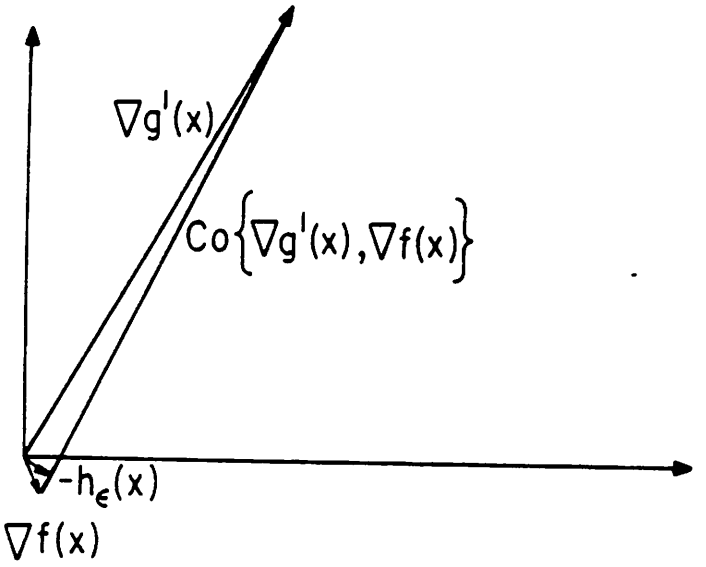

Figure 2. An example of the influence of bad scaling on the search direction calculation.

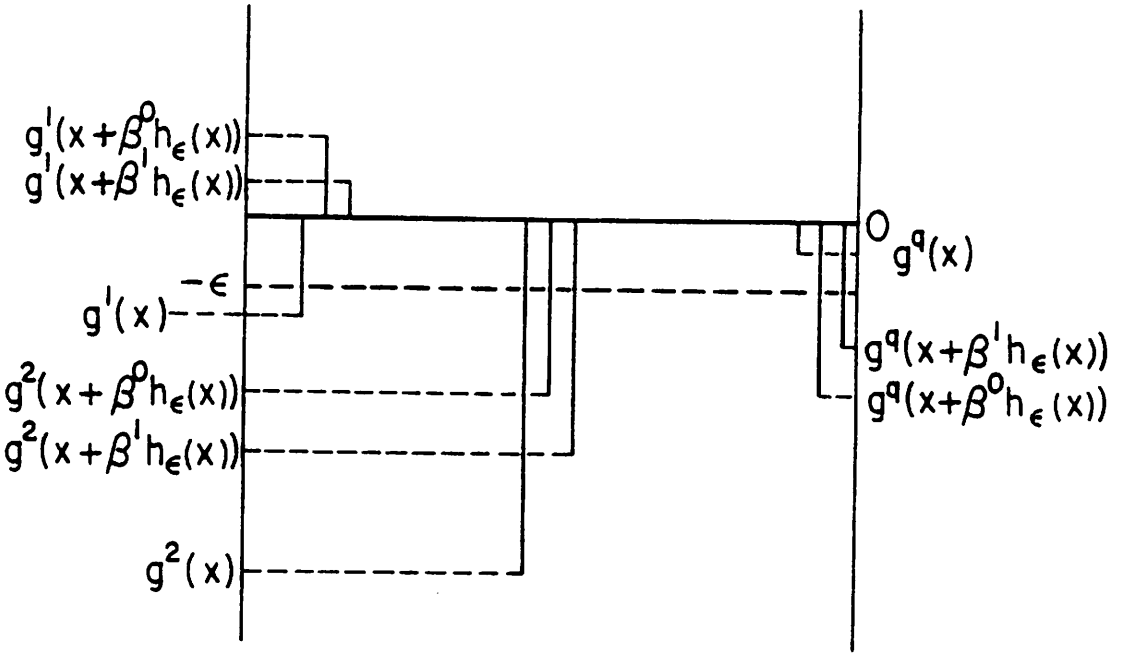

Figure 3. The barchart of the constraints.